Lecture outline

• Dimensionality reduction
  – SVD/PCA
  – CUR decompositions

• Nearest-neighbor search in low dimensions
  – kd-trees
Datasets in the form of matrices

We are given \( n \) objects and \( d \) features describing the objects.
(Each object has \( d \) numeric values describing it.)

**Dataset**
An **n-by-d** matrix \( A \), \( A_{ij} \) shows the “**importance**” of feature \( j \) for object \( i \).
Every row of \( A \) represents an object.

**Goal**
1. **Understand** the structure of the data, e.g., the underlying process generating the data.
2. **Reduce the number of features** representing the data
Market basket matrices

\[ A_{ij} = \text{quantity of } j\text{-th product purchased by the } i\text{-th customer} \]

Find a subset of the products that characterize customer behavior
Social-network matrices

\( A \)

\( A_{ij} = \) participation of the \( i \)-th user in the \( j \)-th group

Find a subset of the groups that accurately clusters social-network users
Document matrices

\[ A_{ij} = \text{frequency of the } j\text{-th term in the } i\text{-th document} \]

Find a subset of the terms that accurately clusters the documents
Recommendation systems

\[ A_{ij} = \text{frequency of the } j\text{-th product is bought by the } i\text{-th customer} \]

Find a subset of the products that accurately describe the behavior of the customers.
The Singular Value Decomposition (SVD)

Data matrices have $n$ rows (one for each object) and $d$ columns (one for each feature).

**Rows:** vectors in a Euclidean space,

Two objects are "close" if the angle between their corresponding vectors is small.

$$A = \begin{pmatrix} 1 & .5 \\ .5 & 1 \end{pmatrix}$$
SVD: Example

Input: 2-d dimensional points

Output:

1st (right) singular vector: direction of maximal variance,

2nd (right) singular vector: direction of maximal variance, after removing the projection of the data along the first singular vector.
Singular values

$\sigma_1$: measures how much of the data variance is explained by the first singular vector.

$\sigma_2$: measures how much of the data variance is explained by the second singular vector.
SVD decomposition

\[
\begin{pmatrix}
A
\end{pmatrix}_{n \times d} =
\begin{pmatrix}
U
\end{pmatrix}_{n \times \ell} \cdot
\begin{pmatrix}
\Sigma
\end{pmatrix}_{\ell \times \ell} \cdot
\begin{pmatrix}
V
\end{pmatrix}_{\ell \times d}^T
\]

\(U \ (V)\): orthogonal matrix containing the left (right) singular vectors of \(A\).
\(\Sigma\): diagonal matrix containing the singular values of \(A\):
\((\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_\ell)\)

Exact computation of the SVD takes \(O(\min\{mn^2, m^2n\})\) time. The top \(k\) left/right singular vectors/values can be computed faster using Lanczos/Arnoldi methods.
SVD and Rank-k approximations

\[ A = U \Sigma V^T \]
Rank-k approximations ($A_k$)

$$
\begin{pmatrix}
    A_k
\end{pmatrix}_{n \times d} =
\begin{pmatrix}
    U_k
\end{pmatrix}_{n \times k} \cdot
\begin{pmatrix}
    \Sigma_k
\end{pmatrix}_{k \times k} \cdot
\begin{pmatrix}
    V_k^T
\end{pmatrix}_{k \times d}
$$

$U_k (V_k)$: orthogonal matrix containing the top $k$ left (right) singular vectors of $A_k$.

$\Sigma_k$: diagonal matrix containing the top $k$ singular values of $A_k$.

$A_k$ is the best approximation of $A$.
SVD as an optimization problem

Find $C$ to minimize:

$$\min_C \left\| A - C X \right\|_F^2$$

$$\|A\|_F^2 = \sum_{i,j} A_{ij}^2$$

Given $C$ it is easy to find $X$ from standard least squares. However, the fact that we can find the optimal $C$ is fascinating!
PCA and SVD

- PCA is SVD done on centered data

- PCA looks for such a direction that the data projected to it has the maximal variance

- PCA/SVD continues by seeking the next direction that is orthogonal to all previously found directions

- All directions are orthogonal
How to compute the PCA

1. Center the data by subtracting the mean of each column
2. Compute the SVD of the centered matrix $A'$ (i.e., find the first $k$ singular values/vectors)
   \[ A' = U\Sigma V^T \]
3. The principal components are the columns of $V$, the coordinates of the data in the basis defined by the principal components are $U\Sigma$
Singular values tell us something about the variance

- The variance in the direction of the $k$-th principal component is given by the corresponding singular value $\sigma_k^2$

- Singular values can be used to estimate how many components to keep

- **Rule of thumb**: keep enough to explain 85% of the variation:

$$\frac{\sum_{j=1}^{k} \sigma_j^2}{\sum_{j=1}^{n} \sigma_j^2} \approx 0.85$$
SVD is the Rolls-Royce and the Swiss Army Knife of Numerical Linear Algebra.”*

*Dianne O’Leary, MMDS ’06
SVD as an optimization problem

Find $C$ to minimize:

$$\min_C \left\| A - C \times X \right\|_F^2$$

$$\left\| A \right\|_F^2 = \sum_{i,j} A_{ij}^2$$

Given $C$ it is easy to find $X$ from standard least squares. However, the fact that we can find the optimal $C$ is fascinating!
The **CX**-decomposition

Find \( \mathbf{C} \) that contains subset of the columns of \( \mathbf{A} \) to minimize:

\[
\min_{\mathbf{C}} \left\| \mathbf{A} - \mathbf{C} \mathbf{X} \right\|_F^2
\]

\[
\left\| \mathbf{A} \right\|_F^2 = \sum_{i,j} A_{ij}^2
\]

Given \( \mathbf{C} \) it is easy to find \( \mathbf{X} \) from standard least squares. However, finding \( \mathbf{C} \) is now hard!!!
Why \textit{CX}-decomposition

• If $A$ is an object-feature matrix, then selecting “representative” columns is equivalent to selecting “representative” features.

• This leads to easier \textit{interpretability}; compare to eigenfeatures, which are linear combinations of all features.
Algorithms for the \textbf{CX} decomposition

- The \textit{SVD-based} algorithm

- The \textit{greedy} algorithm

- The \textit{k-means-based} algorithm
Algorithms for the $CX$ decomposition

• The **SVD-based** algorithm
  – Do SVD first
  – Map $k$ columns of $A$ to the left singular vectors

• The **greedy** algorithm
  – Greedily pick $k$ columns of $A$ that minimize the error

• The **k-means-based** algorithm
  – Find $k$ centers (by clustering the columns)
  – Map the $k$ centers to columns of $A$
Discussion on the CX decomposition

• The vectors in C are not orthogonal – they do not define a space

• It maintains the sparsity of the data
Nearest Neighbour in low dimensions
Definition

• Given: a set $X$ of $n$ points in $\mathbb{R}^d$
• Nearest neighbor: for any query point $q \in \mathbb{R}^d$
  return the point $x \in X$ minimizing $L_p(x,q)$
Motivation

• **Learning**: Nearest neighbor rule

• **Databases**: Retrieval

• Donald Knuth in vol.3 of *The Art of Computer Programming* called it the post-office problem, referring to the application of assigning a resident to the nearest-post office
Nearest neighbor rule
MNIST dataset “2”
Methods for computing NN

• Linear scan: $O(nd)$ time

• This is pretty much all what is known for exact algorithms with theoretical guarantees

• In practice:
  – *kd-trees* work “well” in “low-medium” dimensions
2-dimensional kd-trees

- A data structure to support range queries in \( \mathbb{R}^2 \)
  - Not the most efficient solution in theory
  - Everyone uses it in practice

- Preprocessing time: \( \mathcal{O}(n \log n) \)
- Space complexity: \( \mathcal{O}(n) \)
- Query time: \( \mathcal{O}(n^{1/2} + k) \)
2-dimensional kd-trees

• Algorithm:
  – Choose x or y coordinate (alternate)
  – Choose the median of the coordinate; this defines a horizontal or vertical line
  – Recurse on both sides

• We get a binary tree:
  – Size $O(n)$
  – Depth $O(\log n)$
  – Construction time $O(n \log n)$
Construction of kd-trees
Construction of kd-trees
Construction of kd-trees
Construction of kd-trees
Construction of kd-trees
The complete kd-tree
Region of node $v$

Region($v$) : the subtree rooted at $v$ stores the points in black dots
Searching in kd-trees

• Range-searching in 2-d
  – Given a set of \( n \) points, build a data structure that for any query rectangle \( R \) reports all point in \( R \)
kd-tree: range queries

- Recursive procedure starting from $v = \text{root}$
- **Search** $(v, R)$
  - If $v$ is a leaf, then report the point stored in $v$ if it lies in $R$
  - Otherwise, if $\text{Reg}(v)$ is contained in $R$, report all points in the subtree$(v)$
  - Otherwise:
    - If $\text{Reg(left}(v))$ intersects $R$, then Search$(\text{left}(v), R)$
    - If $\text{Reg(right}(v))$ intersects $R$, then Search$(\text{right}(v), R)$
Query time analysis

• We will show that **Search** takes at most $O(n^{1/2} + P)$ time, where $P$ is the number of reported points
  
  – The total time needed to report all points in all sub-trees is $O(P)$
  
  – We just need to bound the number of nodes $v$ such that $\text{region}(v)$ intersects $R$ but is not contained in $R$ (i.e., boundary of $R$ intersects the boundary of $\text{region}(v)$)
  
  – **gross overestimation**: bound the number of $\text{region}(v)$ which are crossed by any of the 4 horizontal/vertical lines
Query time (Cont’d)

- **Q(n):** max number of regions in an n-point kd-tree intersecting a (say, vertical) line?

  If $\ell$ intersects **region**(v) (due to vertical line splitting), then after two levels it intersects 2 regions (due to 2 vertical splitting lines)

  The number of regions intersecting $\ell$ is $Q(n)=2+2Q(n/4) \Rightarrow Q(n)=(n^{1/2})$
d-dimensional kd-trees

- A data structure to support range queries in $\mathbb{R}^d$
- Preprocessing time: $O(n \log n)$
- Space complexity: $O(n)$
- Query time: $O(n^{1-1/d} + k)$
Construction of the $d$-dimensional \textit{kd-trees}

- The construction algorithm is similar as in $2$-\textit{d}
- At the root we split the set of points into two subsets of same size by a hyperplane vertical to $x_1$-axis
- At the children of the root, the partition is based on the second coordinate: $x_2$-coordinate
- At depth $d$, we start all over again by partitioning on the first coordinate
- The recursion stops until there is only one point left, which is stored as a leaf